# Data Mining Project on Insurance Classification

**Highest Public Score:** 0.56971.

**Group Kaggle ID:** Baseline Model

**Group member:** Ziao Chen (zchen149) worked on code,report

Jinghan Tang (jinghan3) worked on code,report

Kaiji Lu(kaijilu3) worked on report,code

## Introduction

The aim of this project is to predict insurance rating for life insurance applicants based on their attributes. We are given a training set of 20000 rows with over a hundred variables including ID, product info, employment info, insurance history, medical history, ect. The label we aim to give is “Response”, ranging from 1 to 8 with orders, which is why this problem is treated as a regression problem. The label number will be rounded to its nearest integer.

There are several reasons why this project is quite challenging. In addition to what we have covered in midterm report: The proportion of missing values are big; Variable with tricky values (eg., the value of “Product_Info_2” contains both alphabet and number, “D3” for example); Big amount of dummy variables such as “Medical_Key” columns, our new challenge for final phase is removing the library usage, the much longer run time we need to shrink. After trying with varies methods, we managed to use random forest regressor to give a better prediction on the testing dataset and scores 0.568 eventually.

## Detailed Project Description:

### *The problem:*

We are given 20000 samples and labels, each with 108 features to train a model. And then we need to use the model to predict the labels of 10000 unlabeled samples. Though the label is discrete, it somehow should still be treated as continuous value, so a regressor is needed.

## *Model used:*

We build a random forest regressor to do the regression. The main idea is quite simple: the random forest consists of several decision trees, and the final prediction should be the average value coming from all the decision trees.

## *Algorithm:*

First of all, we read in the data from both training.csv and testing.csv. For the missing values, we just replace it with -1. We also used pandas pacakge's factorize method to encode the string type feature.

Then, we randomly pick up samples from the original training dataset, and use those subsets of original dataset to train each decision tree. The single decision tree training algorithm can be concluded as below:

```
def buildtree(subset, dep)
    create new node
    find the best split point of the subset

    split the subset into S1,S2

    update node's split point and feature
    node->left child = buildtree(S1, dep+1)
    node->right child = buildtree(S2, dep+1)
    return node
enddef
```

The most essential part is how to find the best split point and feature. The assumption is, the smaller $Variance(label\ of\ the\ subset) \cdot Size(\ subset\ )$ is, (confirmed by

https://stats.stackexchange.com/questions/190014/why-does-a-regression-tree-not-split-based-on-variance ), the better the split point is.

So the process of find the best split point is:

```
def find split point(subset)
    calculate initial_S = variance(subset)*length(subset)
    randomly pick up features
    best = INF
    for F of all the picked up features:
        for  V of all the value of subset[F]
            split subset by V into S1 and S2
            S = variance(S1)*length(S1) + variance(S2)*length(S2)
            if S < best, update the best split point to V and feature
```

```
 to F
    if best - S < Improve threshold
        return None
    else
        return the best split point and the feature
enddef
```

To get the predicted value from single decision tree, we only need to traverse the decision tree. The algorithm is:

```
def get value(node, input):
    if num of samples of node <= 2:
        return average of the labels
    if input[ node->feature ]  <  node->split point:
        return get value( node->left child, input)
    else
        return get value( node->right child, input)
enddef
```

Finally, we'll use the average of all the predicted results from all the decision trees. We add a linear regression to find the relationship between the random forest's predicted value and label, but it only improves a little.

The main idea about why the random forest can work is that, it uses subsets of features and data points to generate a number of very different decision trees. Some of the trees might be bad, but after we take average of the predicted results from all the trees, such situation can be avoid.

## *Relation to prior work:*

We used completely different method, comparing with our prior work. The reason is we found that we did not have that much time to complete the XGBoost algorithm in time. Also, we remove all the data preprocessing tricks, as we found it useless for the random forest method.

Though, we still use a linear regression between the random forest's predicted value and label, which is similar to our prior work.

## *Parameters and assumptions:*

Maxdepth of decision tree: 100
Number of trees: 80
Sample size for each decision tree: 20% of the original size.
Features picked up: 10

Improvement threshold: 1e-7

We assume that 10 features are enough for the algorithm, considering sklearn package uses sqrt( num of features ) as the limit. We also assume that there is a linear relationship between the predicted value and the actual label, which is why we use a linear regressor after using random forest.

# Final result

We achieved 0.56971 public score on kaggle, which is far more better than random 1~8 baseline model which ends up with 0.

### *Potential Improvements*

May use multithread to improve the speed of traning.

May use OOB score to improve the accuracy.

# Citation:


https://blog.csdn.net/jiede1/article/details/78245597 for the code framework

https://blog.csdn.net/haimengao/article/details/49615955 for the algorithm design

https://www.kaggle.com/thomascleberg/ordinal-regression for the data preprocessing idea and code

https://stats.stackexchange.com/questions/190014/why-does-a-regression-tree-not-split-based-on-variance for the covariance assumption